NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

CAVES - COMPUTER-AIDED VEHICLE EMBARKATION SYSTEM

by

John Michael Byzewski

June 1985

Thesis Advisor: Larry Williamson

Approved for public release; distribution is unlimited.

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* CAVES - Computer-aided Vehicle Embarkation System		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis June 1985
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)* John Michael Byzewski		8. CONTRACT OR GRANT NUMBER*(s)*
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		12. REPORT DATE June 1985
		13. NUMBER OF PAGES 74
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)*
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution is unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Pallet loading
Vehicle loading
Cutting-stock
Template-layout
Knapsack

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

In this thesis the two-dimensional vehicle loading problem is considered: that is, the problem of loading a rectangular deck of size L by W of a ship, drawing from a set of n vehicles. The objective is to maximize the area covered on the deck by the vehicles loaded. A heuristic algorithm is employed to solve the two-dimensional loading problem. A computer-aided vehicle embarkation system (CAVES) is developed to assist embarkation personnel to load vehicles on board a ship. Caves provides the

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

20. (Cont.)

Embarkation Officer the flexibility and portability needed to make real time decisions about vehicle load plans.

Approved for public release; distribution is unlimited.

CAVES - Computer-aided Vehicle Embarkation System

by

John M. Byzewski
Captain, United States Marine Corps
B.S., United States Naval Academy, 1979

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED MATHEMATICS

from the

NAVAL POSTGRADUATE SCHOOL
JUNE 1985

ABSTRACT

In this thesis the two-dimensional vehicle loading problem is considered: that is, the problem of loading a rectangular deck of size L by W of a ship, drawing from a set of n vehicles. The objective is to maximize the area covered on the deck by the vehicles loaded. A heuristic algorithm is employed to solve the two-dimensional loading problem. A computer-aided vehicle embarkation system (CAVES) is developed using a menu driven micro-computer program designed to assist embarkation personnel to load vehicles on board a ship. CAVES provides the Embarkation Officer the flexibility and portability needed to make real time decisions about vehicle load plans.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUND

While embarkation techniques may have changed some over the last 20 years, developing load diagrams still remains a very tedious and manual chore for the Marine Corps Embarkation Officer. Even with the updating from the Mechanized Embarkation Data System (MEDS) to the Standard Embarkation Management System (SEMS) in 1981, the Embarkation Officer still has the task of manually creating load diagrams using paper templates. The Air Force has developed and uses a computer loading system for airplanes called the Deployable Mobility Execution System (DMES); yet, there is no computer loading system currently being used to load Navy ships. In an attempt to fill this void a computer-aided vehicle embarkation system (CAVES) was developed.

B. THE NEED FOR COMPUTERIZATION

Generating load diagrams is somewhat routine. Nevertheless it draws on the experience and skill of the Marine Corps Embarkation Officer to determine the load pattern which yields the best utilization of a ships space, taking into account the tactical requirements. This task is laborious and very time consuming. With the advanced computer technology of today, the archaic use of paper

templates can be updated by a computer system. Thus, using a computer to automatically generate vehicle load diagrams has considerable practical value to the United States Marine Corps.

C. THE MICRO-COMPUTER

The flexibility and portability needed by the Marine Corps Embarkation Officer to make real time decisions about vehicle load plans emphasizes the need for computerization. The micro-computer, with its small physical dimensions and relatively low cost when compared to a mainframe computer, can tackle problems of considerable complexity. It has several advantages over the mainframe machines. The two main advantages are the portability of the micro-computer and the interaction capable between the user and the computer [Ref. 1]. What the micro-computer makes possible is the combination of the speed of a computer together with the skill and experience of the embarkation personnel, in an interactive manner. With the Marine Corps constant deployment on Navy ships around the world, the micro-computer can be there ready to aid embarkation personnel in developing load plans as well as serving as a database management tool, for loading the various vehicles, cargo, and personnel.

D. MARINE CORPS EMBARKATION

Marine Corps embarkation is defined to be the loading of Marines with their supplies and equipment onto ships and/or aircraft. Marine Corps embarkation includes the loading of pallets, personnel, vehicles, and other equipment; however the loading of vehicles is the only topic discussed in this thesis. While the loading of vehicles onto Navy ships is the primary concern, loading can be divided into two types: administrative and combat. Administrative loading emphasizes maximum use of cargo space without regard for tactical considerations. It presumes that the initial destination is a marshalling area where troops and cargo may be discharged free of enemy interdiction. Equipment and supplies must be unloaded before they can be used. Combat loading is the stowage of a vessel so that the equipment needed for a landing attack may be rapidly unloaded in a needed priority. Primary emphasis is placed on tactical considerations rather than the economic use of the ship's space. Marines seldom load ships administratively; therefore, combat loading is the primary consideration of the load algorithm. [Ref. 2]

E. VEHICLE STOWAGE PLANNING

Size, shape, weight, unload priority, and serial grouping are a few of the many details that are considered when planning the stowage of vehicles. Vehicles and cargo

are often loaded in the same compartment; however, vehicles normally have a higher priority for unloading. Therefore, space available for cargo cannot be accurately determined until the vehicle stowage planning is complete. The following specific rules apply when planning vehicle stowage:

1. Each vehicle occupies deck space of specific size and shape.

2. Overhead hatches must be large enough and/or ramp clearances must be sufficient to allow passage of vehicles.

3. If unloading is accomplished by boom/crane or by helicopter, these apparatuses must have sufficient capacity to lift fully loaded vehicles.

4. Each vehicle is positioned on the ship in such a manner as to ensure that it can be unloaded in accordance with its assigned priority number.

5. A marriage, a towed vehicle and its prime mover, must be stowed in the same compartment to ensure that they are not separated during debarkation.

6. Stowage must be planned so that the vehicles can be moved to the ramp, access doors or the space under the overhead hatch square without excessive maneuvering.

7. No vehicle may be stowed athwartships. Vehicles are stowed fore and aft to preclude loosening of lashing caused by the side-to-side movement (roll) of the ship.

8. A broken stowage factor of .8 is applied to the deck loading area to determine the available vehicle stowage area.

F. VEHICLE LOAD PROBLEM

Given n number of vehicles of different sizes, shapes, and weights, the vehicle load problem loads these vehicles

onto a ship's deck of size L by W, accounting for the length, width, height, weight, priority number and marriage constraints of the vehicles. The objective is to maximize the area covered on the deck by the vehicles being loaded. In addition to the above constraints there are a number of independent variables that are considered when loading vehicles on board a ship. The variables of importance are as follows:

1. Clearance between vehicles.
2. Rules for loading vehicles with trailers or towed loads.
3. Rules for loading non-rectangular shaped objects.

Each constraint will be discussed in turn.

Even though the vehicle loading problem is thought of as only a two-dimensional problem (no stacking involved), the height of the vehicle plays a key role. Obviously, if the vehicle exceeds the height restrictions it will not fit in the alloted deck space. Though the weight is listed as a constraint and is a required input for CAVES, the weight distribution of the vehicles loaded on board the ship was not addressed in this thesis. Except for the LKA, the vehicle's weight distribution should not present any loading problems for Navy amphibious ships.

Given that a vehicle meets the height requirements, the primary consideration used in the loading of vehicles is the priority number. The priority number represents the order in

which the vehicles will go ashore. The ordering process is such that the lowest numbered vehicle has the highest priority and must be landed ashore first. When the Embarkation Officer is placing vehicle templates on the deck diagram to configure the load, the highest priority vehicle is placed first. This is continued in numerical order with the rest of the vehicle templates. The loading algorithm in CAVES loads the ship's deck in exactly this manner. This process which is described in Chapter IV is best used for well deck operative ships (e.g. LPD, LSD). When the ship is being physically loaded, the first-in last-out concept is utilized. Therefore, since the vehicle with priority number one is to be landed ashore first, it is physically loaded last on board the ship.

A vehicle marriage, will always have sequential priority numbers. Therefore, it is desirable to be able to load the towed vehicle immediately behind its designated prime mover. Sometimes, because of the space limitations, this is impossible. Therefore, the towed vehicle must be loaded beside its prime mover. Trailers or similiar towed vehicles can be loaded in one of two ways:

1. Remain attached to the prime mover.
2. Disengage the lunette of the trailer and push the tow bar beneath the prime mover to decrease total length.

The minimum clearance between vehicles is one foot. An additional six inches is added to the length and width of

each vehicle to obtain the minimum clearance. The one foot spacing between vehicles allows the vehicles to be tied-down to the deck.

The loading of irregularly shaped towed loads such as artillery pieces presents a special problem. Because of their triangular shape they tend to occupy more space and require more manuevering by hand. Since CAVES considers only rectangularly shaped vehicles, the artillery pieces could be loaded in pairs making a roughly rectangular shape.

II. OBJECTIVES AND SCOPE

The objectives of this study were to review the existing loading algorithms and develop a user-friendly computer program to help embarkation personnel load vehicles on board a ship. CAVES was designed as a menu driven computer program to make it as simple as possible to utilize. The intent was to design a computer program that a user with little or no embarkation experience could use. The system was developed specifically for use on a micro-computer and presupposes that the user has some knowledge of operating a micro-computer and that all necessary data for the vehicles to be loaded are available. It is assumed that all vehicles are rectangular, no irregular shapes are used, and the ship's deck where the vehicles are to be loaded is rectangular. It is also assumed that the area of the ship's deck where the vehicles are being loaded is clear of any obstacles or non-stowable areas.

Neither the weight distribution nor the center of gravity restrictions are addressed by the load algorithm. However, these restrictions could be included by modifying the load algorithm.

III. REVIEW OF EXISTING ALGORITHMS

The interest in two-dimensional allocation type problems has grown over the past thirty years. This is in part due to the important role these problems play in computer-aided and computer-automated design applications, particularly those related to sheet metal fabrication, garment making, template layout and circuit layout [Ref. 3]. With the appearance of large mainframe computers and the development of computer oriented optimization techniques such as linear and dynamic programming, the interest has continued to grow. In this chapter a survey of some of the available literature concerning methods used to solve these two-dimensional problems is presented. First of interest is the work of Gilmore and Gomory [Ref. 4] and their developement of knapsack functions. Next, two related problems, the Cutting Stock and Template-Layout problems, are introduced. Finally the Loading problem is presented.

A. THE KNAPSACK PROBLEM

The problem of cutting a one-dimensional object (e.g. a length of some material) into smaller pieces - each piece having a given length and value - in such a way as to maximize the total value of pieces cut is the "knapsack

problem". Mathematically the knapsack function F(x) is defined for lengths $l_1,\ldots,l_m$ of given values $II_1,\ldots,II_m$ by the equation:

$$F(x) = \max\{Z_1 II_1 + \ldots + Z_m II_m \; ; \; l_1 Z_1 + \ldots + l_m Z_m \leq x\}$$
$$\{Z_i \geq 0,\ Z_i \text{ integer}\}$$

where II_i and l_i are given constants, $i = 1,\ldots m$. Here the problem is one of fitting lengths l_i into a box of length x. The knapsack problem has been examined by a number of authors, and methods for its solution have been proposed using either dynamic programming [Ref. 5] or tree search techniques [Ref. 6]. The attention here is motivated by many practical problems that can be formulated as knapsack problems, one typical case being the vehicle loading problem [Ref. 7]. Given rectangles of dimensions (l_i, w_i), $i = 1,\ldots,m$ that have nonnegative values $II_1,\ldots,II_m$ associated with them a two-dimensional knapsack function G is defined as follows:

$$G(x,y) = \max\{II_1 Z_1 + \ldots + II_m Z_m\}$$

where $Z_1,\ldots,Z_m$ are nonnegative integers such that there exists a way of dividing a large rectangle (x,y) into Z_i rectangles (l_i, w_i), $i = 1,\ldots m$. [Ref. 4, pp. 1045-1046]

B. THE CUTTING STOCK PROBLEM

The two-dimensional cutting stock problem requires cutting a large rectangle into smaller rectangular pieces of specified sizes and values so as to minimize the total waste. Hahn [Ref. 8] investigates the cutting stock problem in the two-dimensional trim problem which has arisen in industries that produce glass, veneer, film, etc. To minimize waste, manufacturers use some material with defective areas. The manufacturers get orders for a certain number of different sizes to be cut out of the available sheets. The problem is now to find the combination of sizes to be cut out of the available sheets which minimizes the waste. The dynamic programming approach used in this optimization, requires that a value be attached to every size. Whenever a size is fitted in, its value is added to the total value of the sizes already used for that sheet. The optimal combination will be the one with the greatest possible value.

Christofides and Whitlock [Ref. 9] use a tree-search algorithm for the two-dimensional cutting problem in which there is a constraint on the maximum number of each type of piece that is to be produced. Their algorithm limits the size of the tree search by imposing necessary conditions for the cutting pattern to be optimal. They use a dynamic programming procedure for the solution of the unconstrained problem.

The above algorithm is also constrained to consider only cutting patterns made by guillotine type cuts. A guillotine type cut is one in which all cuts go from one edge of the rectangle to be cut, to the opposite edge. Figure 2(a) shows a possible cutting pattern using only guillotine cuts while Figure 2(b) shows a cutting pattern that could not be produced using guillotine cuts.

Steudel [Ref. 10] uses a heuristic algorithm employing dynamic programming to solve the two-dimensional cutting stock problem in which all the small rectangles are of the same dimensions and nonguillotine cuts are allowed.

C. THE TEMPLATE-LAYOUT PROBLEM

The template-layout problem is characterized by the requirement for cutting two-dimensional shapes or rectangles out of large sheets in an optimum manner without making an exhaustive search of all possible arrangements [Ref. 11]. This problem is closely related though distinct from, the cutting stock problem. In the template-layout problem the objective is to obtain as many pieces as possible from a fixed number of sheets; where as in the cutting stock problem the objective is to satisfy a fixed demand with a minimum number of sheets.

In 1970 Haims and Freeman [REf. 12] developed an algorithm for solving the template-layout problem. A dynamic programming approach was applied, assuming that there was an

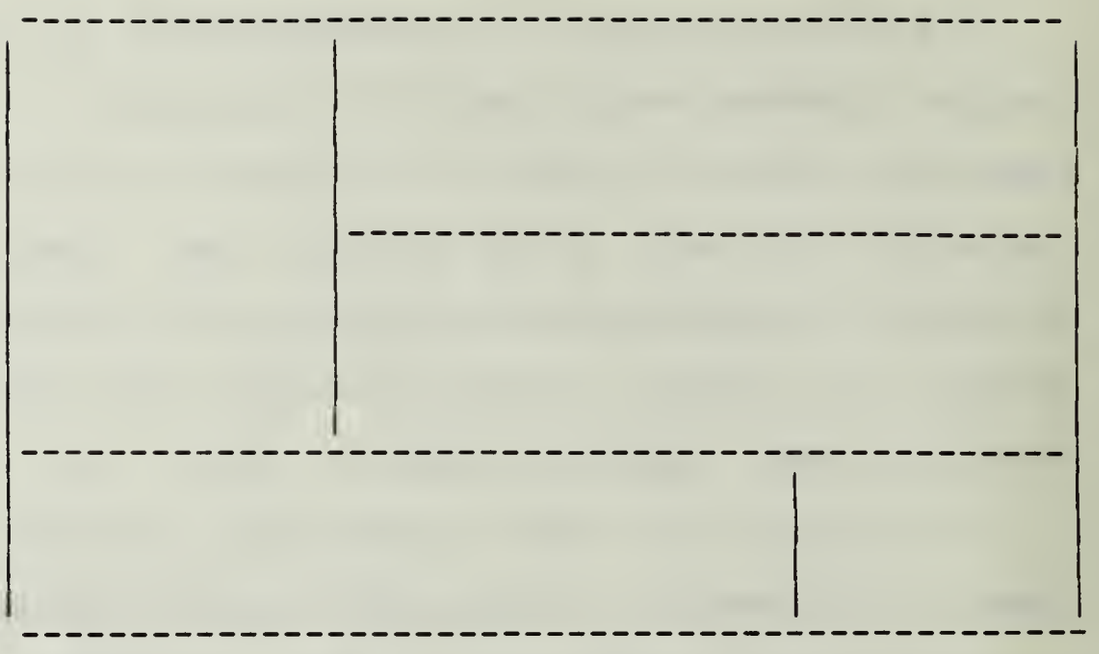

Figure 1a. Cutting pattern made with guillotine cuts.

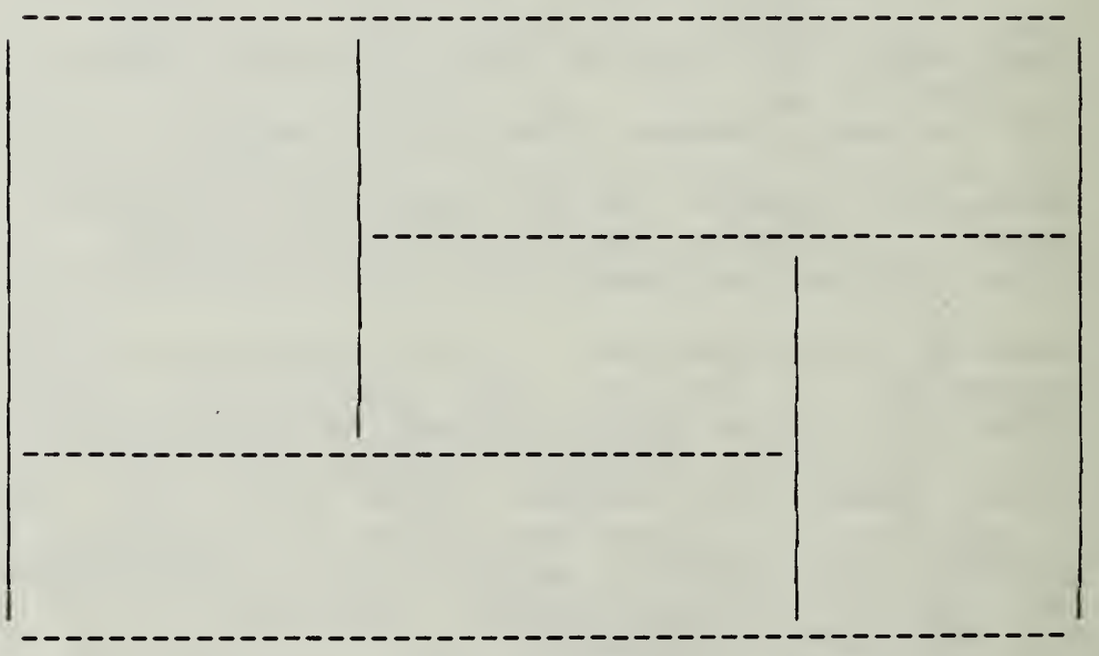

Figure 1b. Cutting pattern infeasible with guillotine cuts.

Figure 1. Cutting Patterns

unlimited supply of each type of rectangle and that the orientation of the rectangles may be either fixed or left unspecified.

D. THE LOADING PROBLEM

There are several types of loading problems. Eilon and Christofides [Ref. 13] discuss one type of loading problem, which is akin to the knapsack problem. This particular type of loading problem is defined as the allocation of given items with known magnitude to boxes with constrained capacity, so as to minimize the number of boxes used. They present two methods for solving this problem:

1. The Zero-One programming method.
2. A heuristic algorithm.

Another type of loading problem is the pallet loading problem. The pallet loading problem can be viewed as a special case of the two-dimensional cutting stock problem where all the small rectangles are of identical dimensions. The problem consists of partitioning a rectangular pallet of length L by W into smaller rectangular areas of length l and width w so as to determine a loading pattern which minimizes the amount of unused pallet area. Another version of the pallet loading problem as viewed by Hodgson [Ref. 14] is the problem of loading a rectangular pallet of size L by W, drawing from a set of n rectangular boxes. The objective here is to maximize the area covered on the pallet

by the boxes loaded. Hodgson's approach was to use a combination of dynamic programming and heuristics. His procedure is in fact a generalization and extension of the Template-Layout problem by Haims and Freeman.

Another version of the loading problem analogous to Hodgson's version is the vehicle loading problem. As evidenced by the above review, the vehicle loading problem is related to the knapsack, cutting stock, and template-layout problems. The vehicle loading algorithm used in this thesis is discussed in the next chapter.

IV. THE LOADING ALGORITHM

A. GENERAL DESCRIPTION

The main function of CAVES is to load vehicles on a ship's deck, minimizing the area used. The second option of CAVES' main menu allows the user to load a ship's deck of specified length, width, and height (See Appendix A for menu option details). CAVES uses a hueristic loading routine to load the vehicles.

The loading algorithm used was incorporated from a thesis written by Nelson [Ref. 15]. The algorithm loads one vehicle at a time by inspecting each of the n vehicles in the vehicle file. Vehicle with priority number one is inspected first, vehicle with priority number two is inspected second, and so forth until all n vehicles are inspected. If the vehicle under inspection can be loaded without violation of one of the constraints discussed in section IF above, the vehicle is loaded. If the current vehicle does violate one of the constraints, it is passed over and the next vehicle in the sequence is inspected and loaded if possible. If none of the vehicles waiting to be loaded can be loaded, the loading process is ended.

The loading procedure requires a decision to be made as to where an additional vehicle may be placed. These locations are defined as "possible" origins. They are

constrained by the algorithm to be either the origin of the ship's deck or one of two positions relative to each of the vehicles previously loaded. The first vehicle being loaded is loaded at the bottom left corner of the deck and thereafter the possible origins are either to the right of or behind the previously loaded vehicle, as shown in Figure 2. The coordinate axis may be thought of in relating the x and y positions. The x axis is the width while the y axis is the length. The dimensions of the vehicle being added are denoted by VEHL, VEHW, and VEHH corresponding to its length, width and height respectively. These positions are selected as possible origins because they limit the possible positions of the next vehicle to a finite, manageable number of locations. Finally, of all the possible origins, a subset of "permissible" origins is defined. This subset of "permissible" origins is determined by deleting from all possible origins those which have already been utilized by loaded vehicles. [Ref. 16]

The order of inspection of those permissible origins is:

1. The x-position ordered from the first vehicle loaded to the last vehicle loaded.
2. The y-position ordered from the first to the last vehicle loaded.

This order of inspection tends to load the deck in rows, always starting from the deck's origin (bottom left corner).

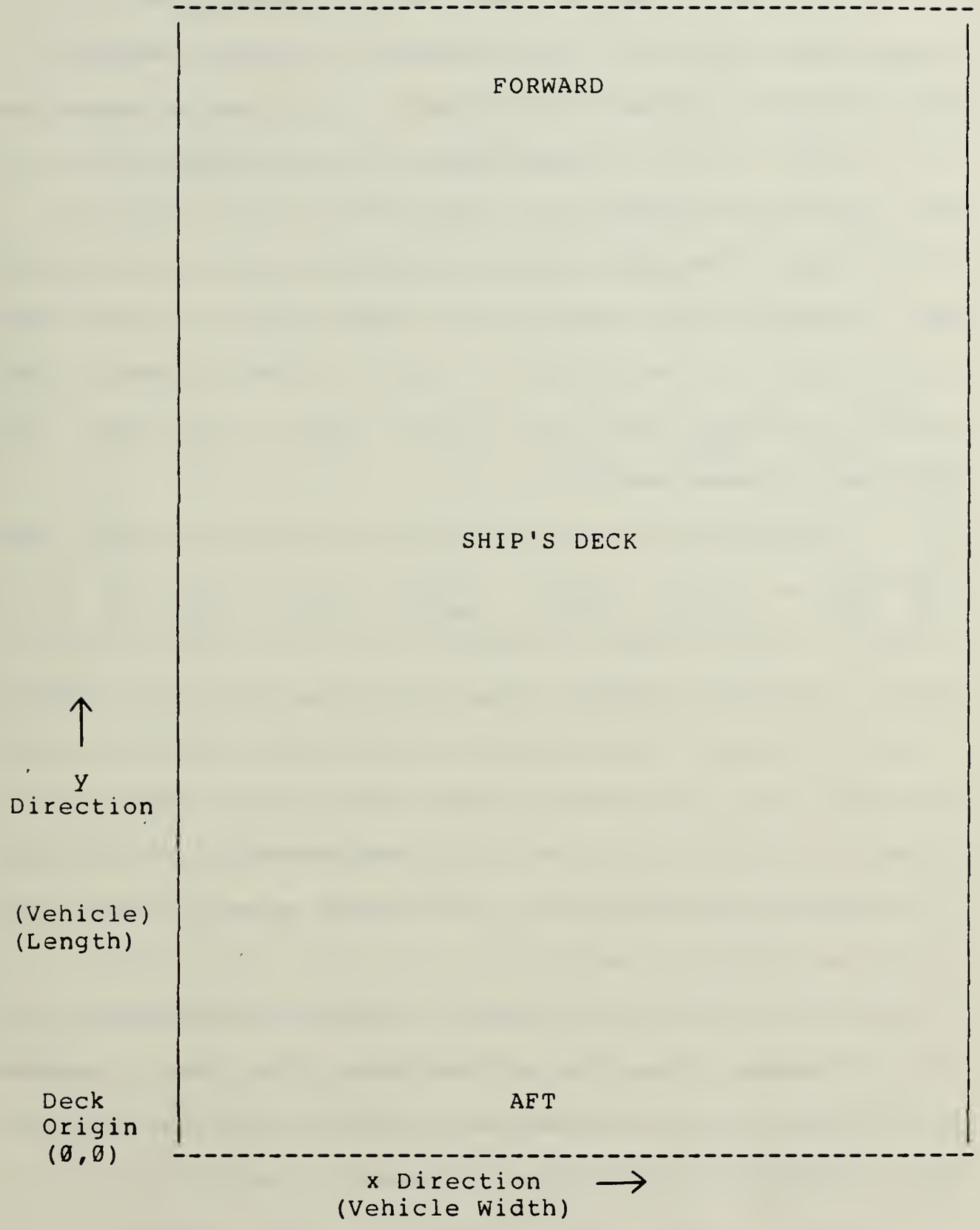

Figure 2. Possible Origins for Placing Vehicles on the Deck.

To determine if a vehicle may be loaded at a given permissible origin, it is necessary to maintain a record of all previously loaded vehicles and their relative positions on the deck. This is accomplished by maintaining a record of the previously loaded vehicles' dimensions in the x and y directions. Thus, feasiblilty is determined by considering the vehicle at a particular permissible origin and determining if the vehicle is wholly contained within the space of the deck and if the vehicle does not intersect any previously loaded vehicle.

A local minimum is obtained by attempting to move each vehicle, as it is loaded, toward the origin of the deck. This is accomplished by determining if the vehicle may be moved along one of the x or y directions toward the deck's origin. Movement is only permitted if the vehicle does not intersect any previously loaded vehicle. The vehicle is moved in one direction at a time and movement is continued in an iterative fashion until no further movement toward the origin is possible. [Ref. 17]

The next section will briefly describe the notation used in the loading algorithm and following that, the algorithm is stated. After the statement of the algorithm, a brief sample problem is solved for illustrative purposes.

B. NOTATION AND DESCRIPTION OF MATRICES

Before stepping through and describing the loading algorithm, notation is briefly discussed and the matrices used in the algorithm are defined.

There are n vehicles to be loaded and their characteristics are contained in matrix D. Matrix D is defined as an (nx6) matrix of vehicles that are to be loaded as follows:

$$D = \begin{pmatrix} PRI\#_1 & VEHL_1 & VEHW_1 & VEHH_1 & VSQFT_1 & WGT_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ PRI\#_n & VEHL_n & VEHW_n & VEHH_n & VSQFT_n & WGT_n \end{pmatrix}$$

where PRI# is the priority number of the vehicle.

VEHL is the length of the vehicle.

VEHW is the width of the vehicle.

VEHH is the height of the vehicle.

VSQFT is the square feet occupied by the vehicle.

WGT is the weight of the vehicle.

Matrix D is used in the algorithm as a device to maintain a record of the vehicles yet to be loaded.

To make a determination of whether a vehicle will fit at a given origin, Matrix R is established and updated with each vehicle that is loaded onto the deck. Matrix R is defined as follows:

$$R = \begin{pmatrix} 1 & X_1 & Y_1 & (X_1+VEHW_1) & (Y_1+VEHL_1) & VEHH_1 & VSQFT_1 & WGT_1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ N_n & X_n & Y_n & (X_n+VEHW_n) & (Y_n+VEHL_n) & VEHH_n & VSQFT_n & WGT_n \end{pmatrix}$$

N_j is the priority number of VEH_j whose length, width and height are $VEHL_j$, $VEHW_j$, and $VEHH_j$ respectively, where $1 \leq j \leq n$, and whose origin is located at coordinates X_j, Y_j.

Thus, the first column of matrix R identifies the priority number of the vehicle, columns two and three identify the vehicle's location on the deck and columns four and five describe the area occupied by the vehicle while column six is the vehicle height. Column seven is somewhat redundant by listing the square footage of the vehicle and column eight lists the vehicle's weight.

Matrix R allows the determination of fit to be made through a series of very fast logic checks. These logic checks are discussed in the next section.

To facilitate the selection of the next origin where the algorithm attempts to load the current vehicle, a logical matrix of possible and permissible origins is established. By an extremely rapid scan of this matrix, defined as Matrix B, the next origin is quickly determined. Matrix B is an (nx2) matrix as follows:

$$B = \begin{pmatrix} XORG_1 & YORG_1 \\ \vdots & \vdots \\ XORG_n & YORG_n \end{pmatrix}$$

Each row of matrix B corresponds to a row in matrix R. The two elements in each row of matrix B correspond to the x-direction and y-direction possible origins associated with each vehicle as described by matrix R. Of all the possible origins, the permissible origins are defined by setting a true value to each element in matrix B which corresponds to the possible origin which is also a permissible origin. It is precisely these permissible origins where attempts are made to load additional vehicles.

C. STATEMENT OF THE ALGORITHM

To load n vehicles onto a ship's deck the following steps are used:

1. Input the length, width and overhead height restriction of the deck.
2. Check to see that all vehicles' heights are within the height limitations.
3. Establish matrix B.
4. Load the first vehicle. Augment matrices R and B with an additional row to represent this vehicle. Adjust matrix B if necessary, by removing an origin from the set of permissible origins.

5. Select the next vehicle in matrix D. If no more vehicles are left, go to step 10.

6. Select the next permissible origin. The next permissible origin corresponds to the next true element in column 1 of matrix B (x-position), followed by the next true element in the second column (y-position). Call the selected origin (ORW,ORL). If all origins have been tried go to step 5.

7a. Determine if the vehicle will fit at this origin. This is determined by the following logic checks:

For the x direction,

(R(j,4) + VEHW) > DECKW or (R(j,3) + VEHL) > DECKL

For the y direction,

(R(j,5) + VEHL) > DECKL or (R(j,2) + VEHW) > DECKW

where $1 \leq j \leq n$. A true condition indicates that the vehicle will not fit at this origin.

7b. Improve the density of loading if possible. Inspect, one at a time, each possible direction of improvement (x,y). Each direction of improvement is found by inspecting the origin (ORW,ORL) under question and each row of matrix R. To determine if improvement is possible in the x direction, the following logic check is made on each row of matrix R:

R(j,2) > (ORW + VEHW) or R(j,3) > (ORL + VEHL) or

R(j,6) < ORL.

A true condition indicates that improvement is not possible at this row in matrix R. A false condition indicates that improvement is possible. The magnitude of improvement is ORW - R(j,5), and is denoted in the algorithm as the variable slack [Ref. 18]. The improvement found over all rows of matrix R is:

$$\min(\text{slack}_1, \ldots, \text{slack}_n, \text{ORW}).$$

To determine improvement in the y direction the logic check is:

R(j,3) > (ORL + VEHL) or R(j,2) > (ORW + VEHW) or

R(j,5) < ORW.

The magnitude of improvement when the logic check is false is ORL - R(j,6).

With each improvement, the origin (ORW,ORL) of the vehicle being loaded is adjusted. The search for improvement is continued until no improvement can be found in any direction.

8. Load the vehicle and adjust matrices R and B with an additional row. Adjust matrix B to preclude any origin that may not be used.

9. Go to 5.

10. All the vehicles have been either loaded or have been attempted to be loaded; terminate the algorithm.

When the algorithm terminates, it does not in general yield the true optimal solution. The vehicle loading problem falls in the category of problems called NP-Hard [Ref. 14 p.176]. Consequently, a truly optimal algorithm is not likely to be found.

D. SAMPLE PROBLEM

In order to illustrate the load algorithm, the following example is presented. There are five vehicles to be loaded as shown in matrix D.

$$D = \begin{pmatrix} 1 & 135 & 64 & 54 & 60 & 2500 \\ 2 & 112 & 61 & 43 & 48 & 610 \\ 3 & 219 & 80 & 74 & 122 & 4648 \\ 4 & 118 & 50 & 50 & 41 & 1000 \\ 5 & 227 & 84 & 91 & 133 & 7300 \end{pmatrix}$$

Step 1. In this example the length, width and overhead height limitations of the ship's deck are 400, 200 and 100 respectively. The units are in inches, except for the vehicle area which is in square feet and the weight is in pounds.

Step 2. Check to see that all the vehicles' heights in matrix D are less than the overhead height limitation.

Step 3. Establish the B matrix of origins.

Step 4. Load the first vehicle in matrix D. This is the vehicle with priority # 1 and dimension of 135x64x54. Augment the R matrix to include this vehicle as follows:

R = (1 0 0 64 135 54 60 2500).

Augment the B matrix and show permissible origins by setting the applicable element to true and remove if necessary any origins from the permissible origins. Thus

B = (T T).

Step 5. Select the next vehicle in the D matrix. This is the vehicle with priority # 2 and dimensions 112x61x43.

Step 6. Select the next permissible origin. Scan matrix B column by column always starting at the top of each column and working down. In this example, element B(1,1) is the next permissible origin. This element translates into an origin of (R(1,4),R(1,3)) or (64,Ø). Denote this as the current (ORW,ORL).

Step 7A. Determine if the vehicle will fit at this origin. This is accomplished by the following logic check of row one of matrix R:

(R(1,4) + VEHW) > DECKW or R(1,3) + VEHL) > DECKL

This equates to:

(64 + 61) > 2ØØ or (Ø + 135) > 4ØØ

which is obviously false becuase 125 is not greater than 2ØØ and 135 is not greater than 4ØØ. Therefore, the vehicle will fit at this origin.

Step 7B. Since this is only the second vehicle improving the density is not possible.

Step 8. Load the vehicle and augment the matrices R and B as follows:

$$R = \begin{pmatrix} 1 & Ø & Ø & 64 & 135 & 54 & 6Ø & 25ØØ \\ 2 & 64 & Ø & 125 & 112 & 43 & 48 & 61Ø \end{pmatrix}$$

$$B = \begin{pmatrix} F & T \\ T & T \end{pmatrix}$$

Step 5. The next vehicle in matrix D is priority # 3 whose dimensions are 219x80x74.

Step 6. The next permissible origin of matrix B is B(2,1). This element translates into an origin of (R(2,4),R(2,3)) or (125,0). Set ORW = 125 and ORL = 0.

Step 7A. Determine if the vehicle will fit at this origin. The following logic check is made:

(125 + 80) > 200 or (0 + 219) > 400

Since the logic check is true for the x direction, the vehicle will not fit at this origin.

Step 6. Select the next permissible origin. Since this vehicle would not fit in the x direction, the next permissible origin is related to the y direction and is B(1,2) which translates to R(1,2),R(1,5) or (0,135).

Step 7A. Determine if the vehicle will fit at this origin. The following logic check is made:

(R(1,5) + VEHL) > DECKL or (R(1,2) + VEHW) > DECKW

which equates to:

(135 + 219) > 400 or (0 + 80) > 200

which is false therefore, the vehicle will fit at this origin.

Step 7B. The density cannot be improved.

Step 8. Load the vehicle and augment the matrices R and B as follows:

$$R = \begin{pmatrix} 1 & 0 & 0 & 64 & 135 & 54 & 60 & 2500 \\ 2 & 64 & 0 & 125 & 112 & 43 & 48 & 610 \\ 3 & 0 & 135 & 80 & 354 & 74 & 122 & 4648 \end{pmatrix}$$

$$B = \begin{pmatrix} F & F \\ T & F \\ T & F \end{pmatrix}$$

In the augmentation of the B matrix, B(2,2) was set to false because if a vehicle were to be loaded at that origin it would intersect the vehicle with priority # 3. B(3,2) was also set to false because the distance from the third vehicle loaded to the ship's aft loading boundry is less than the length of the shortest vehicle, 46 as compared to 112.

Step 5. The next vehicle in matrix D is priority # 4 with dimension 118x50x50.

Step 6. Since no vehicle was loaded at B(2,1) or origin (125,0), select this as the next permissible origin.

Step 7. This vehicle will fit and improvement is not possible.

Step 8. Load the vehicle and adjust R and B matrices as follows:

$$R = \begin{pmatrix} 1 & 0 & 0 & 64 & 135 & 54 & 60 & 2500 \\ 2 & 64 & 0 & 125 & 112 & 43 & 48 & 610 \\ 3 & 0 & 135 & 80 & 354 & 74 & 122 & 4648 \\ 4 & 125 & 0 & 175 & 118 & 50 & 41 & 1000 \end{pmatrix}$$

$$B = \begin{pmatrix} F & F \\ F & F \\ T & F \\ F & T \end{pmatrix}$$

Step 5. The next and last vehicle in matrix D is priority # 5 with dimension 227x84x91.

Step 6. Because the distance from the fourth vehicle loaded to the ship's right bulkhead is less than the width of the smallest vehicle, 25 as compared to 43, B(4,1) was set to false. B(3,1) is the next permissible origin. This equates to (R(3,4),R(3,3) or (80,135).

Step 7. The vehicle will fit. Check to see if improvement is possible. For illustration purposes the y direction will be shown. To determine if improvement is possible in the y direction, the following logic check is made:

R(4,3) > (ORL + VEHL) or R(4,2) > (ORW + VEHW) or

R(4,4) < ORW

which equates to:

Ø > (135 + 227) or 125 > (8Ø + 84) or 175 < 8Ø

Since this logic check is false, improvement in the y direction is possible. The magnitude of improvement is ORL - R(4,6) or 135 - 118 which equals 17. No improvement is possible in the x direction therefore, the origin of the vehicle is now ORW = 8Ø, ORL = 117.

Step 8. Load the vehicle and augment the R and B matrices as follows:

$$R = \begin{pmatrix} 1 & Ø & Ø & 64 & 135 & 54 & 6Ø & 25ØØ \\ 2 & 64 & Ø & 125 & 112 & 43 & 48 & 61Ø \\ 3 & Ø & 135 & 8Ø & 354 & 74 & 122 & 4648 \\ 4 & 125 & Ø & 175 & 118 & 5Ø & 41 & 1ØØØ \\ 5 & 8Ø & 118 & 164 & 345 & 91 & 133 & 73ØØ \end{pmatrix}$$

$$B = \begin{pmatrix} F & F \\ F & F \\ F & F \\ F & F \\ F & F \end{pmatrix}$$

Step 1Ø. Matrix R now shows the ship's deck as it is loaded; terminate the algorithm.

CAVES has produced a results matrix of where the vehicles should be loaded. Figure 3 shows the deck diagram of where each vehicle would be placed. Using this deck diagram the Embarkation Officer would then physically load the ship in reverse order using the first-in last-out concept.

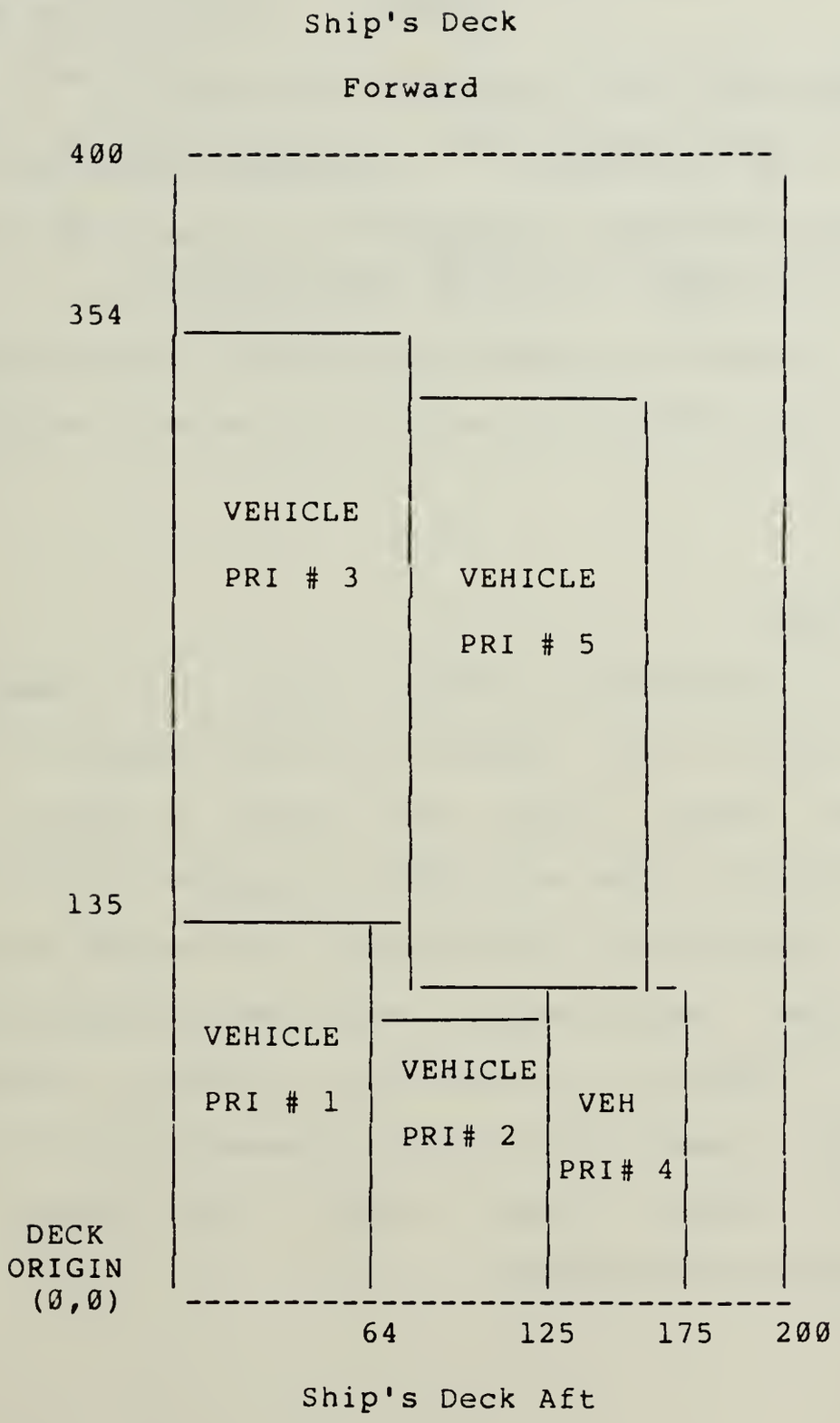

Figure 3. Example Problem

V. SUMMARY

A review of the available literature on loading algorithms was conducted. A heuristic algorithm was incorporated from Nelson and modified for use in the loading algorithm for CAVES. Intended primarily as an aid for Embarkation personnel, CAVES accomplished its objective by simulating the tactical loading of vehicles on a rectangular deck.

A. CONCLUSIONS

CAVES is an adequate start at trying to computerize vehicle loading. Much remains to be done before a fully computerized loading system can become a reality. The "Expert" loading system will need to consider not only the vehicles, but pallets, non-stowable areas on the deck, and non-rectangular loading areas. While vehicle graphics are presented in Appendix B, a graphical system to produce a vehicle load diagram showing the placement of the loaded vehicles is needed to make CAVES a more useable and effective embarkation system.

B. RECOMMENDATIONS

1. That research and development of a fully automated computer embarkation system for the Navy and the Marine Corps be continued.

2. That CAVES be modified to include the following:

   a) obstacles or non-stowable areas in the load algorithm,

   b) use of a graphical system to produce load diagrams showing the placement of the loaded vehicles.

APPENDIX A

CAVES

A. STRUCTURE OF CAVES

CAVES was developed on and designed to run on a microcomputer. The particular machine chosen was the IBM Personal Computer configured with two DS/DD disk drives and 372k bytes of RAM memory. Written in PASCAL MT+, CAVES is comprised of four overlays: 1. EMBARK; 2. IOVEH; 3. VEHICLE; and 4. VHELP. EMBARK is the main module or root overlay for the three other overlays. The menu (option list) for EMBARK contains four options:

1. CREATE, UPDATE OR DISPLAY A FILE - sends the user to the overlay IOVEH in order to create, update or display the vehicle file.

2. LOAD SHIP - sends the user to the overlay VEHICLE in order to use the load algorithm.

3. INSTRUCTIONS FOR USE OF CAVES - sends the user to the overlay VHELP.

4. QUIT/EXIT SYSTEM - terminates the CAVES program.

The IOVEH overlay is the module responsible for the keyboard input of files and printer output of files. IOVEH acts as the database manager for CAVES. It allows the user to create a new vehicle file or update the existing vehicle file. It also allows the user to display the current file, including the results of the loading algorithm. Within the

IOVEH overlay, procedures Writeveh, and Readveh are the work horses. The Writeveh procedure stores the vehicle data and in conjunction with the Readveh procedure allows the user to edit the vehicle file. Procedure Fileio is the editing procedure which calls the Procedures Edit and Display. The procedure Edit displays menus, opens up the files, and calls the above mentioned procedures to edit existing files. The procedure Display gives a screen display of the files, while Procedure Hardcopy is used to printout the vehicle and result files. The menu (option list) for IOVEH contains three options:

1) CREATE OR EDIT VEHICLE FILE- puts the user in the file editor mode and allows the user to select one of three options:

   a) CREATE NEW VEHICLE FILE

   b) EDIT OLD VEHICLE FILE

   e) EXIT

2) SCREEN DISPLAY OR HARDCOPY OUTPUT - puts the user in the file display mode and allows the user to select one of the following five options:

   a) SCREEN DISPLAY OF VEHICLE FILE

   b) SCREEN DISPLAY OF RESULTS FILE

   e) HARDCOPY OUTPUT OF VEHICLE FILE

   f) HARDCOPY OUTPUT OF RESULTS FILE

   g) EXIT

3) EXIT - ends the input/output session and exits to the main program.

The VEHICLE overlay prompts the user for the length, width and overhead height restriction of the ship's deck where the vehicles are to be loaded. It also prompts the user for the number of vehicles from the vehicle file that are to be loaded. An end of loading message appears on the screen when the loading algorithm is finished.

The overlay VHELP is a basic guide for the first time user of CAVES.

B. DESCRIPTION OF CRT AND KEYBOARD UTILITIES

Incorporated throughout CAVES is an input filtering system. which allows the user to type in only the proper input. For example, to select an option from the main program menu one of four inputs is required, either A,B,C or E. If anything else is typed in, the bell will sound indicating improper input.

The utility routines for the filtering system and cursor control are found in the Module CRTLIB. There are over twenty different utility routines that are found in CRTLIB but only ten are used frequently. They are as follows:

1) Procedure Crtinit - To use the utilities in Module CRTLIB the procedure Crtinit must be called at the beginning of the main program.

Crtinit initializes the arrays CRTINFO and PREFIXED so that their values can be used by the utilities in Module CRTLIB.

2) Procedure Gotoxy(X,Y:INTEGER) - Places the cursor at vertical line number X and horizontal line number Y.

3) Procedure Promptat(Y:INTEGER;S:STRING) - Places the prompt after the string S on line number Y.

4) Procedure Clearscreen - Clears the screen and places the prompt at the (Ø,Ø) position.

5) Procedure Clearit(I:INTEGER) - Clears the screen of everything from line I to the bottom of the screen.

6) Function Getchar(OKSET:SETCHAR): CHAR; - Performs the task of reading a character from the keyboard. The input to Getchar is a variable set of characters, called OKSET in Getchar's declaration. When a character is inputted from the keyboard, Getchar verifies that it is in OKSET. If it is, this character becomes the value of the Function Getchar and is used in the procedure calling Getchar; if it is not, a beep is sounded and the process is repeated until a character in OKSET is entered.

7) Function Yes: BOOLEAN - Uses Getchar to check if the input is a 'Y' and if so sets its value to true.

8) Procedure Whead(S:STRING) - Whead will center, print and underline a string S. It is primarily used for headings.

9) Procedure Intread(VAR K:INTEGER) - Intread is used to read an integer between -32768 and +32767. The characters

are filtered, put into a string, checked for the proper range, and then converted to an integer.

10) Procedure Spacebar - Used as a manual stopping mechanism. The message "Press Spacebar" is displayed on the screen to the user and then Getchar is called to insure the spacebar input.

It is the use of the above routines within CAVES that makes CAVES a user friendly computer program.

APPENDIX B

VEHICLE GRAPHICS

To enhance and make CAVES a more useable product, a set of vehicle graghics were created. As shown in Figure 4 each vehicle is a re-creation of actual templates used by embarkation personnel to load ships. While not actually incorporated into CAVES, the vehicle graphics are a needed aid in helping embarkation personnel load ships. Used in conjuction with a graphical loading system and CAVES, the vehicle graphics would "print" a picture of the results from the loading algorithm.

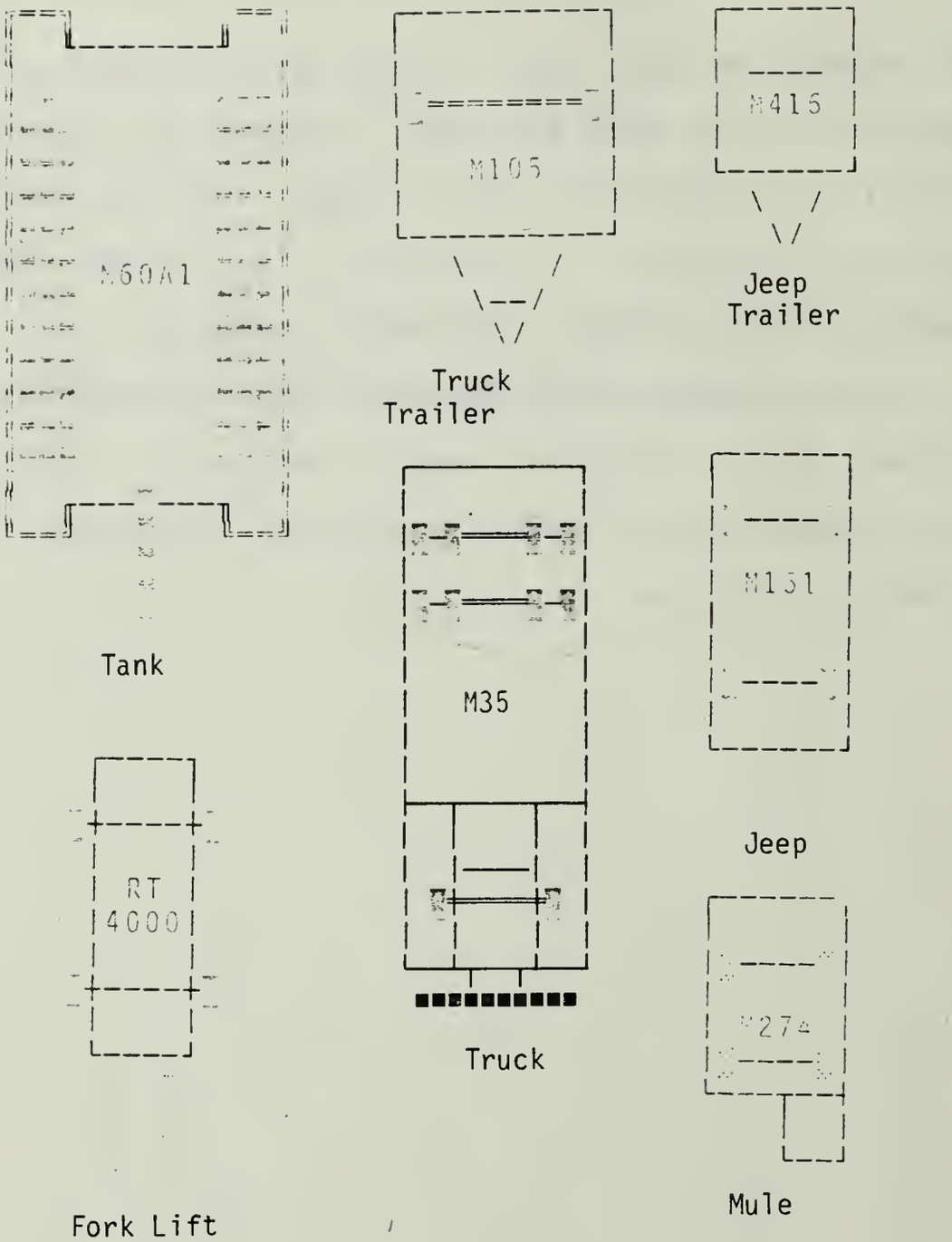

Figure 4. Vehicle Graphics

APPENDIX C

COMPUTER PROGRAM

A. PROGRAM EMBARK

```
PROGRAM EMBARK;

CONST BELL = 07;
      RTN = 13;
      BSP = 8;
      MAXCOLR = 8;
      MAXCOLD = 6;

TYPE
     STRING16 = STRING[16];
     CRTCOMMAND = (ERASEOS,ERASEOL,UP,DOWN,RIGHT,LEFT,
                   LEADIN,TIME,FCOLOR,BCOLOR,REVIDON,
                   REVIDOFF,INTENON,INTENOFF,BLINKON,
                   BLINKOFF);
     SETOFCHAR = SET OF CHAR;
     PTR = ^INTEGER;
     CPMOPERATION = (COLDBOOT,WARMBOOT,CONSTAT,CONIN,
                     CONOUT,LIST,PUNOUT,RDRIN,HOME,
                     SELDSK,SETTRK,SETSEC,SETDMA,
                     DSKREAD,DSKWRITE);
     STRING40 = STRING[40];
     STRING60 = STRING[60];
     NUMVEH = 1..100;
     MATCOLR = 1..8;
     MATCOLB = 1..2;
     MATCOLD = 1..6;
     MATRIX1 = ARRAY[NUMVEH,MATCOLR] OF INTEGER;
     MATRIX2 = ARRAY[NUMVEH,MATCOLB] OF BOOLEAN;
     MATRIX3 = ARRAY[NUMVEH,MATCOLD] OF INTEGER;
 VAR I,J,IONDIM,IOM,ROWDIMB: INTEGER;
     ROBFLAG,AOIFLAG,IOBFLAG,QUITFLAG: BOOLEAN;
     R: MATRIX1;
     D: MATRIX3;
     B: MATRIX2;
     SELECT: CHAR;
     OKSET: SETOFCHAR;
     SBLASTX,SBLASTY: EXTERNAL INTEGER;

(* EXTERNAL PROCEDURES AND FUNCTIONS *)

    EXTERNAL PROCEDURE CRTINIT;
```

```
    EXTERNAL PROCEDURE CRT(C:CRTCOMMAND);

    EXTERNAL PROCEDURE GOTOXY(X,Y:INTEGER);

    EXTERNAL PROCEDURE PROMPTAT(Y:INTEGER;S:STRING);

    EXTERNAL PROCEDURE CLEARSCREEN;

    EXTERNAL PROCEDURE CLEARIT(I:INTEGER);

    EXTERNAL FUNCTION GETCHAR(OKSET:SETOFCHAR): CHAR;

    EXTERNAL FUNCTION YES: BOOLEAN;

    EXTERNAL PROCEDURE GETSTRING(VAR S:STRING;MAXLEN: INTEGER);

    EXTERNAL PROCEDURE WAIT;

    EXTERNAL PROCEDURE WHEAD(S:STRING);

    EXTERNAL PROCEDURE INTREAD(VAR K:INTEGER);

    EXTERNAL PROCEDURE SPACEBAR;

    EXTERNAL [3] PROCEDURE FILEIO;

    EXTERNAL [19] PROCEDURE LOAD;

    EXTERNAL [23] PROCEDURE HELP;

(* END OF EXTERNAL DECLARATIONS *)

BEGIN (* EMBARK *)

  CRTINIT;
  QUITFLAG := FALSE;
  AOIFLAG := FALSE;
  IOBFLAG := FALSE;
  ROBFLAG := FALSE;
  REPEAT
    CLEARSCREEN;
    WHEAD(' COMPUTER AIDED VEHICLE EMBARKATION SYSTEM ');
    GOTOXY(0,3);
    WRITELN(' ':5,'A ','CREATE,UPDATE OR DISPLAY A FILE');
    WRITELN;
    WRITELN(' ':5,'B ','LOAD SHIP');
    WRITELN;
    WRITELN(' ':5,'C ','INSTRUCTIONS FOR USE OF CAVES');
    WRITELN;
    WRITELN(' ':5,'Q ','QUIT/EXIT SYSTEM');
    WRITELN;
```

```
  WRITE(' ':5,'SELECT ONE : ');
  OKSET := ['A'..'C','Q'];
  SELECT := GETCHAR(OKSET);
  CLEARSCREEN;
  IF SELECT = 'Q' THEN
  BEGIN
    PROMPTAT(10,'DO YOU REALLY WANT TO QUIT? TYPE Y FOR
                 YES, N FOR NO');
    IF YES THEN QUITFLAG := TRUE;
    CLEARSCREEN;
  END
  ELSE
    CASE SELECT OF
       'A': FILEIO;
       'B': LOAD;
       'C': HELP;
    END;
UNTIL QUITFLAG;
END. (* EMBARK *)
```

B. OVERLAY IOVEH

```
MODULE OVERLAY3;

CONST BELL = 07;
      RTN = 13;
      BSP = 8;
      MAXCOLR = 8;
      MAXCOLD = 6;

TYPE
     STRING16 = STRING[16];
     CRTCOMMAND = (ERASEOS,ERASEOL,UP,DOWN,RIGHT,LEFT,LEADIN,
                   TIME, FCOLOR,BCOLOR,REVIDON,REVIDOFF,INTENON,
                   INTENOFF, BLINKON,BLINKOFF);
     SETOFCHAR = SET OF CHAR;
     PTR = ^INTEGER;
     CPMOPERATION = (COLDBOOT,WARMBOOT,CONSTAT,CONIN,CONOUT,
                     LIST, PUNOUT,RDRIN,HOME,SELDSK,SETTRK,
                     SETSEC,SETDMA, DSKREAD,DSKWRITE);
     STRING40 = STRING[40];
     STRING60 = STRING[60];
TYPE NUMVEH = 1..100;
     MATCOLR = 1..8;
     MATCOLB = 1..2;
     MATCOLD = 1..6;
     MATRIX1 = ARRAY[NUMVEH,MATCOLR] OF INTEGER;
     MATRIX2 = ARRAY[NUMVEH,MATCOLB] OF BOOLEAN;
     MATRIX3 = ARRAY[NUMVEH,MATCOLD] OF INTEGER;
     VECTOR  = ARRAY[MATCOLD] OF INTEGER;
     VECTOR1 = ARRAY[MATCOLR] OF INTEGER;
VAR
    IOM,IONDIM,ROWDIMB: EXTERNAL INTEGER;
    ROBFLAG,AOIFLAG,IOBFLAG: EXTERNAL BOOLEAN;
    R: EXTERNAL MATRIX1;
    D: EXTERNAL MATRIX3;
    F: FILE OF VECTOR;
    F1: FILE OF VECTOR1;

(* EXTERNAL PROCEDURES AND FUNCTIONS *)

    EXTERNAL PROCEDURE CRTINIT;

    EXTERNAL PROCEDURE CRT(C:CRTCOMMAND);

    EXTERNAL PROCEDURE GOTOXY(X,Y:INTEGER);

    EXTERNAL PROCEDURE PROMPTAT(Y:INTEGER;S:STRING);

    EXTERNAL PROCEDURE CLEARSCREEN;
```

```
    EXTERNAL PROCEDURE CLEARIT(I:INTEGER);

    EXTERNAL FUNCTION GETCHAR(OKSET:SETOFCHAR): CHAR;

    EXTERNAL FUNCTION YES: BOOLEAN;

    EXTERNAL PROCEDURE GETSTRING(VAR S:STRING;MAXLEN: INTEGER);

    EXTERNAL PROCEDURE WAIT;

    EXTERNAL PROCEDURE WHEAD(S:STRING);

    EXTERNAL PROCEDURE INTREAD(VAR K:INTEGER);

    EXTERNAL PROCEDURE SPACEBAR;
(* END OF EXTERNAL DECLARATIONS *)

PROCEDURE  PMEN(I:INTEGER;C:CHAR;S:STRING); (* CREATES MENU
                                              FOR OVERLAY*)

BEGIN (* PMEN *)
  GOTOXY(0,I);
  WRITELN(C:3,' ':3,S);
END; (* PMEN*)

PROCEDURE HEAD(A:STRING;J:INTEGER);  (* PRINTS HEADING *)
VAR I:INTEGER;
BEGIN (* HEAD *)
  I := (80 - LENGTH(A)) DIV 2;
  GOTOXY(I,J);
  WRITELN(A);
END; (* HEAD *)


PROCEDURE FILEIO;
VAR   QUITFLAG: BOOLEAN;
      CHOICE: CHAR;
      OKSET: SETOFCHAR;

    PROCEDURE WRITEVEH(I:INTEGER); (* WRITES CAVES DATA TO
                                      FILE VEH.DAT *)

    LABEL 100;
    VAR IOR,K,L,J: INTEGER;
        WVEFLAG: BOOLEAN;

    BEGIN (* WRITEVEH *)
    HEAD('CAVES WRITE FILE MODE',0);
    WVEFLAG := FALSE;
    REPEAT
```

```
    ASSIGN(F,'B:VEH.DAT');  (* CREATES NEW VEHICLE
                                    FILE VEH.DAT *)
    RESET(F);
    100: (* CONTINUE *)
    CLEARIT(1);
    GOTOXY(1,2);
    K := 3;
    WRITELN('ROW ',I,' OF THE VEHICLE FILE IS: ');
    WRITELN;
    WRITELN('PRI #      LENGTH    WIDTH     HEIGHT
                                      SQFT      WEIGHT');
    GOTOXY(3,6);
    FOR L := 1 TO MAXCOLD DO
    BEGIN
      INTREAD(J);
      F^[L] := J;
      K := K + 10;
      GOTOXY(K,6);
    END;
    SEEKWRITE(F,I);
    CLOSE(F,IOR);
    WRITELN;
    WRITELN('ROW # ',I,' HAS BEEN WRITTEN TO THE VEHICLE
                                                FILE ');
    ASSIGN(F,'B:VEH.DAT');
    RESET(F);
    CLEARIT(1);
    WRITELN;
    WRITELN('ROW ',I,' OF THE VEHICLE FILE IS: ');
    WRITELN('PRI #      LENGTH    WIDTH     HEIGHT
                                    SQFT      WEIGHT');
    WRITELN;
    SEEKREAD(F,I);
    WRITE('   ',F^[1],'    ',F^[2],'   ',F^[3],'        ');
    WRITELN(F^[4],'     ',F^[5],'      ',F^[6]);
    CLOSE(F,IOR);
    PROMPTAT(7,'IS THIS CORRECT? TYPE Y FOR YES N FOR NO ');
    IF NOT YES THEN GOTO 100;
    WVEFLAG := TRUE;
  UNTIL WVEFLAG;
END; (* WRITEVEH *)

PROCEDURE READVEH(I:INTEGER);(* READS/EDITS CAVES DATA
                                        FROM VEH.DAT *)
  VAR IOR,K,L,J: INTEGER;
  RVEFLAG: BOOLEAN;

  BEGIN (* READVEH *)
    HEAD('CAVES FILE EDIT MODE',0);
    RVEFLAG := FALSE;
    REPEAT
```

```
    ASSIGN(F,'B:VEH.DAT');  (* OPENS EXISTING VEHICLE
                                        FILE VEH.DAT *)
    RESET(F);
    CLEARIT(1);
    GOTOXY(1,2);
    WRITELN('ROW ',I,' OF THE VEHICLE FILE IS: ');
    WRITELN;
    WRITELN('PRI #      LENGTH    WIDTH     HEIGHT
                            SQFT      WEIGHT');
    SEEKREAD(F,I);
    WRITE('  ',F^[1],'    ',F^[2],'    ',F^[3],'       ');
    WRITELN(F^[4],'        ',F^[5],'         ',F^[6]);
    CLOSE(F,IOR);
    PROMPTAT(7,'IS THIS CORRECT? TYPE Y FOR YES,
                                          N FOR NO.');
    IF NOT YES THEN
    BEGIN
      ASSIGN(F,'B:VEH.DAT');
      RESET(F);
      CLEARIT(1);
      GOTOXY(1,2);
      K := 3;
      WRITELN('ROW ',I,' OF THE VEHICLE FILE IS: ');
      WRITELN;
      WRITELN('PRI #      LENGTH    WIDTH     HEIGHT
                               SQFT      WEIGHT');
      GOTOXY(3,6);
      FOR L := 1 TO MAXCOLD DO
      BEGIN
        INTREAD(J);
        F^[L] := J;
        K := K + 10;
        GOTOXY(K,6);
      END;
      SEEKWRITE(F,I);
      CLOSE(F,IOR);
      WRITELN;
      WRITELN('ROW # ',I,' HAS BEEN WRITTEN TO THE VEHICLE
                                                FILE ');
    END
    ELSE
      RVEFLAG := TRUE;
  UNTIL RVEFLAG;
END; (* READVEH *)


   PROCEDURE EDIT;
   VAR EDQUIT: BOOLEAN;
       EDCHOICE: CHAR;

       PROCEDURE AENTER(AORB:CHAR);
```

```
  VAR IOR,K,I,J,COLDIM,ROWDIM: INTEGER;
      MANAME: STRING[9];
      AENFLAG: BOOLEAN;
  BEGIN (* AENTER *)
    CLEARSCREEN;
    CASE AORB OF
      'A': BEGIN
             ASSIGN(F,'B:VEH.DAT');
             REWRITE(F);
             CLOSE(F,IOR);
             MANAME := 'VEHICLE';
           END;
    END;
    AENFLAG := FALSE;
    REPEAT
      CLEARIT(1);
      GOTOXY(1,2);
      WRITE('WHAT ROW OF THE ',MANAME,' FILE DO YOU
                                        WANT TO ENTER ');
      INTREAD(I);
      CASE AORB OF
        'A': WRITEVEH(I);
      END;
      PROMPTAT(10,'DO YOU WISH TO CONTINUE? TYPE
                                             Y OR N');
      IF NOT YES THEN AENFLAG := TRUE;
    UNTIL AENFLAG;
  END; (* AENTER *)

  PROCEDURE AEDIT(AORB:CHAR);
  VAR   I,J,COLDIM,ROWDIM: INTEGER;
        MANAME: STRING[9];
        AEDFLAG: BOOLEAN;
  BEGIN (* AEDIT *)
    CLEARSCREEN;
    AEDFLAG := FALSE;
    REPEAT
      CLEARIT(1);
      GOTOXY(1,2);
      WRITE('ENTER ROW # YOU WISH TO EDIT ');
      INTREAD(I);
      CASE AORB OF
        'A': READVEH(I);
      END;
      PROMPTAT(10,'DO YOU WISH TO CONTINUE TYPE
                                           Y OR N ');
      IF NOT YES THEN AEDFLAG := TRUE;
    UNTIL AEDFLAG;
  END; (* AEDIT *)

BEGIN (* EDIT *)
```

```
          CLEARSCREEN;
          WHEAD(' CAVES FILE EDITOR MENU ');
          EDQUIT := FALSE;
          REPEAT
            CLEARIT(1);
            PMEN(2,'A','CREATE NEW VEHICLE FILE');
            PMEN(4,'B','EDIT OLD VEHICLE FILE');
            PMEN(6,'E','EXIT');
            GOTOXY(2,10);
            WRITE('SELECT ONE : ');
            OKSET := ['A','B','E'];
            EDCHOICE := GETCHAR(OKSET);
            CASE EDCHOICE OF
              'A': AENTER('B');
              'B': AEDIT('B');
              'E': EDQUIT := TRUE;
            END;
          UNTIL EDQUIT;
        END; (* EDIT *)

PROCEDURE DISPLAY;
VAR   DISQUIT: BOOLEAN;
DISCHOICE: CHAR;

   PROCEDURE DDISPLA(AORB:CHAR);
   LABEL 250,275;
   VAR I,J,K,IOR,ROWDIM,COUNTER: INTEGER;
       CH: CHAR;
       CFLAG: BOOLEAN;
       MANAME: STRING[9];
   BEGIN (* DDISPLA *)
     CLEARSCREEN;
     HEAD('CAVES SCREEN DISPLAY MODE',0);
     CASE AORB OF
       'A': MANAME := 'VEHICLE';
       'B': MANAME := ' RESULT';
     END;
     GOTOXY(1,2);
     WRITELN('THE ',MANAME,'S OF THE ',MANAME,' FILE ARE : ');
     WRITELN;
     ROWDIM := 15;
     COUNTER := 1;
     CASE AORB OF
         'A': BEGIN
                ASSIGN(F,'B:VEH.DAT');
                RESET(F);
                WRITELN;
                WRITE('PRI #     LENGTH     WIDTH     HEIGHT
                                                    SQFT');
                WRITELN('      WEIGHT');
                250: (* CONTINUE *)
```

```
              FOR I := COUNTER TO ROWDIM DO
              BEGIN
                SEEKREAD(F,I);
                WRITE('   ',F^[1],'   ',F^[2],'   ',F^[3]);
                WRITELN('   ',F^[4],'   ',F^[5],'   ',F^[6]);
                ROWDIM := ROWDIM + 15;
                COUNTER := COUNTER + 15;
              END;
              PROMPTAT(22,'DO YOU WISH TO CONTINUE TYPE
                                                  Y OR N');
              IF YES THEN
              BEGIN
                GOTOXY(1,6);
                GOTO 250;
              END;
              CLOSE(F,IOR);
              SPACEBAR;
            END;
      'B': BEGIN
              ASSIGN(F1,'B:RESULTS.DAT');
              RESET(F1);
              WRITELN;
              WRITE('PRI #  X      Y      X+VEHW        ');
              WRITELN('Y+VEHL    HEIGHT    SQFT     WEIGHT');
              275: (* CONTINUE *)
              FOR I := 1 TO ROWDIM DO
              BEGIN
                SEEKREAD(F1,I);
                WRITE('  ',F1^[1],'     ',F1^[2],'         ');
                WRITE(F1^[3],'      ',F1^[4],'     ',F1^[5]);
                WRITELN(' ',F1^[6],'  ',F1^[7],'   ',F1^[8]);
                ROWDIM := ROWDIM + 15;
                COUNTER := COUNTER +15;
              END;
              PROMPTAT(22,'DO YOU WISH TO CONTINUE TYPE
                                                  Y OR N');
              IF YES THEN
              BEGIN
                GOTOXY(1,6);
                GOTO 275;
              END;
              CLOSE(F1,IOR);
              SPACEBAR;
            END;
    END;
END; (* DDISPLA *)

PROCEDURE HCOPY(AORB:CHAR);
VAR CH: CHAR;
    CFLAG: BOOLEAN;
    MANAME: STRING[9];
```

```
      CNUM,I: INTEGER;
    BEGIN (* HCOPY *)
      CASE AORB OF
        'A': CNUM := 1;
        'B': CNUM := 2;
      END;
      CLEARIT(1);
      HARDCOPY(CNUM);
    END; (* HCOPY *)

BEGIN (* DISPLAY *)
  DISQUIT := FALSE;
  REPEAT
    CLEARSCREEN;
    WHEAD(' CAVES FILE DISPLAY MENU ');
    PMEN(2,'A','SCREEN DISPLAY OF VEHICLE FILE');
    PMEN(4,'B','SCREEN DISPLAY OF RESULTS FILE');
    PMEN(6,'C','HARDCOPY OUTPUT OF VEHICLE FILE');
    PMEN(8,'D','HARDCOPY OUTPUT OF RESULTS FILE');
    PMEN(10,'E','EXIT');
    GOTOXY(2,14);
    WRITE('SELECT ONE : ');
    OKSET := ['A'..'E'];
    DISCHOICE := GETCHAR(OKSET);
    CASE DISCHOICE OF
      'A': DDISPLA('A');
      'B': DDISPLA('B');
      'C': HCOPY('A');
      'D': HCOPY('B');
      'E': DISQUIT := TRUE;
    END;
  UNTIL DISQUIT;
END; (* DISPLAY *)

BEGIN (* FILEIO *)
  CLEARSCREEN;
  QUITFLAG := FALSE;
  REPEAT
    CLEARSCREEN;
    WHEAD(' COMPUTER AIDED VEHICLE EMBARKATION SYSTEM ');
    PMEN(2,'A','CREATE OR EDIT VEHICLE FILE');
    PMEN(4,'B','SCREEN DISPLAY OR HARDCOPY OUTPUT');
    PMEN(6,'E','EXIT');
    GOTOXY(2,8);
    WRITE('SELECT ONE : ');
    OKSET := ['A','B','E'];
    CHOICE := GETCHAR(OKSET);
    CASE CHOICE OF
      'A': EDIT;
      'B': DISPLAY;
      'E': QUITFLAG := TRUE;
```

```
    END;
  UNTIL QUITFLAG;
END; (* FILEIO *)

PROCEDURE HARDCOPY(HNUM: INTEGER);
VAR F: TEXT;
    CH: CHAR;
    PRFLAG: BOOLEAN;
    DIM,RESULT: INTEGER;

    PROCEDURE SETPRINT;
    VAR CH: CHAR;
        FTRIES: INTEGER;
    BEGIN (* SETPRINT *)
      PRFLAG := FALSE;
      FTRIES := Ø;
      REPEAT
        ASSIGN(F,'LST:');
        REWRITE(F);
        IF IORESULT = 255 THEN
        BEGIN
          FTRIES := FTRIES + 1;
          IF FTRIES <= 2 THEN
          BEGIN
            WRITELN(' PUT PRINTER ON LINE ');
            SPACEBAR;
          END;
        END
        ELSE
          PRFLAG := TRUE;
      UNTIL PRFLAG OR (FTRIES>2);
    END; (* SETPRINT *)

    PROCEDURE PRTFILE(AORB: CHAR;DIM: INTEGER);
    CONST MAXCOLR = 8;
          MAXCOLD = 6;
    VAR I,J: INTEGER;
        MANAME: STRING[9];
    BEGIN (* PRTFILE *)
      CASE AORB OF
        'A': MANAME := 'VEHICLE';
        'B': MANAME := 'RESULTS';
      END;
      WRITELN(F);
      WRITELN(F,'THE ',MANAME,' FILE');
      WRITELN(F);
      FOR I := 1 TO DIM DO
      BEGIN
        CASE AORB OF
          'A': FOR J := 1 TO MAXCOLD DO
               BEGIN
```

```
                WRITE(F,D[I,J]);
                WRITE(F,' ');
              END;
        'B': FOR J := 1 TO MAXCOLR DO
              BEGIN
                WRITE(F,R[I,J]);
                WRITE(F,' ');
              END;
      END;
      WRITELN(F);
    END;
  END; (* PRTFILE *)

BEGIN (* HARDCOPY *)
  WRITELN;
  WRITE('HOW MANY ROWS OF THE FILE DO YOU WISH PRINTED?  ');
  INTREAD(DIM);
  SPACEBAR;
  CLEARSCREEN;
  SETPRINT;
  IF PRFLAG THEN
  BEGIN
    CASE HNUM OF
      1: PRTFILE('A',DIM);
      2: PRTFILE('B',DIM);
    END;
    CLOSE(F,RESULT);
  END;
END; (* HARDCOPY *)

MODEND.
```

C. OVERLAY VEHICLE

```
MODULE OVERLAY19;

TYPE
     STRING16 = STRING[16];
     CRTCOMMAND = (ERASEOS,ERASEOL,UP,DOWN,RIGHT,LEFT,LEADIN,
                   TIME,FCOLOR,BCOLOR,REVIDON,REVIDOFF,
                   INTENON,INTENOFF,BLINKON,BLINKOFF);
     SETOFCHAR = SET OF CHAR;
     PTR = ^INTEGER;
     CPMOPERATION = (COLDBOOT,WARMBOOT,CONSTAT,CONIN,CONOUT,
                     LIST,PUNOUT,RDRIN,HOME,SELDSK,SETTRK,
                     SETSEC,SETDMA,DSKREAD,DSKWRITE);
     STRING40 = STRING[40];
     NUMVEH = 1..100;
     MATCOLR = 1..8;
     MATCOLB = 1..2;
     MATCOLD = 1..6;
     MATRIX1 = ARRAY[NUMVEH,MATCOLR] OF INTEGER;
     MATRIX2 = ARRAY[NUMVEH,MATCOLB] OF BOOLEAN;
     MATRIX3 = ARRAY[NUMVEH,MATCOLD] OF INTEGER;
     VECTOR  = ARRAY[MATCOLD] OF INTEGER;
     VECTOR1 = ARRAY[MATCOLR] OF INTEGER;

VAR
    R: EXTERNAL MATRIX1;
    B: EXTERNAL MATRIX2;
    D: EXTERNAL MATRIX3;
    IBPR,IBPC,IAP,ICP,NL,NWORG,NLORG,NZORG,VEHH,SMLW,SMLL,
    AREA,ORL,ORW,ORZ,ORLDL,ORWDW,ORZDH,SLACK,XYZ,VEHL,VEHW,
    SMLZ,DECKL,DECKW,VSQFT,WGT: INTEGER;
    ALLGON,CHANGE,CHECKD,SOMCHG: BOOLEAN;
    SBLASTX,SBLASTY,ROWDIMB: EXTERNAL INTEGER;
    F: FILE OF VECTOR;
    F1: FILE OF VECTOR1;

(* EXTERNAL PROCEDURES AND FUNCTIONS *)

    EXTERNAL PROCEDURE INTREAD(VAR K:INTEGER);

    EXTERNAL PROCEDURE SPACEBAR;


PROCEDURE LOAD;
LABEL 500,600,605,610,615,620,640,645,700,705,710,711,
      722,725,730,736,800,835,999,9999;
CONST MAXCOLD = 6;
      MAXCOLR = 8;
VAR   J,I,IBPR,IBPC,IAP,ICP,NL,NXORG,NYORG,NZORG,VEHW,VEHH,
```

```
      SMLX,SMLY,SMLZ,AREA,ORX,ORY,ORZ,ORXDL,ORYDW,ORZDH,
      SLACK,XYZ,VEHL,IOR,DECKL,DECKH,DECKW,VSQFT,WGT: INTEGER;
      ALLGON,CHANGE,CHECKD,SOMCHG: BOOLEAN;

    PROCEDURE B4LOAD;
    VAR I,J,K,IOR: INTEGER;

    BEGIN (* B4LOAD *)
      WRITE('ENTER NUMBER OF VEHICLES TO BE LOADED ');
      INTREAD(NL);
      WRITELN;
      WRITE('THE NUMBER OF VEHICLES TO BE LOADED IS ',NL);
      WRITELN;
      WRITE('ENTER DECK LENGTH ');
      INTREAD(DECKL);
      WRITELN;
      WRITELN('THE DECK LENGTH IS ',DECKL);
      WRITE('ENTER DECK WIDTH ');
      INTREAD(DECKW);
      WRITELN;
      WRITELN('THE DECK WIDTH IS ',DECKW);
      WRITELN;
      WRITE('ENTER OVERHEAD HEIGHT LIMIT ');
      INTREAD(DECKH);
      ASSIGN(F,'B:VEH.DAT');
      RESET(F);
      FOR I := 1 TO NL DO
      BEGIN
        SEEKREAD(F,I);      (* TRANSFER FROM VEH.DAT TO D
                                                  MATRIX *)
        FOR J := 1 TO MAXCOLD DO
        D[I,J] := F^[J];
        FOR K := 2 TO 3 DO
        D[I,K] := D[I,K] + 6;
      END;
      CLOSE(F,IOR);
      WRITELN;
      WRITELN('AFTER THE 6 INCH ADJUSTMENT THE LOAD
                                              MATRIX IS:  ');
      FOR I := 1 TO NL DO
      BEGIN
        FOR J := 1 TO 6 DO
        BEGIN
          WRITE(D[I,J]);
          WRITE(' ');
        END;
      WRITELN;
      END;
      SPACEBAR;
END;
```

```
BEGIN (* LOAD *)
  B4LOAD;
  WRITELN('NL IN LOAD IS ',NL);
  FOR I := 1 TO NL DO
  BEGIN
    IF (D[I,4] >= DECKH) THEN
  DIR A:
  BEGIN
    WRITELN('THE LOADING PROCESS HAS BEEN STOPPED BECAUSE');
    WRITELN;
    WRITELN('VEHICLE PRI # ',I,' EXCEEDS HEIGHT LIMITS');
    SPACEBAR;
    GOTO 9999;
  END;
END;
SMLL := 10000;
SMLW := 10000;
(* SELECT THE SMALLEST PERMISSABLE MARGINS IN THE
                                X AND Y DIRECTIONS *)
FOR J := 1 TO NL DO
BEGIN
  IF (D[J,2] < SMLL) THEN SMLL := D[J,2];
  IF (D[J,3] < SMLW) THEN SMLW := D[J,3];
END;
IAP := 0;
B[1,1] := TRUE;
B[1,2] := TRUE;
AREA := 0;
ICP := 0;
500:
ICP := ICP + 1;
IF (ICP > NL) THEN GOTO 999;
NWORG := 1;
NLORG := 1;
IF ((D[ICP,2] > DECKL) OR (D[ICP,3] > DECKW)) THEN
BEGIN
  WRITELN('THE LOADING PROCESS HAS BEEN STOPPED BECAUSE');
  WRITELN;
  WRITELN('VEHICLE PRI # ',D[I,1],' EXCEEDS BOUNDARY
         LIMITATIONS');
  SPACEBAR;
  GOTO 9999;
END;
VEHL := D[ICP,2];
VEHW := D[ICP,3];
VEHH := D[ICP,4];
VSQFT := D[ICP,5];
WGT := D[ICP,6];
600:  (* CONTINUE *)
IF (IAP = 0) THEN     (* IAP=0, IMPLIES 1ST
                              PASS THROUGH ALGORITHM *)
```

```
BEGIN
  ORW := 0;
  ORL := 0;
  IBPR := 1;
  IBPC := 1;
  GOTO 800;
END
ELSE
  GOTO 605;
605: (* CONTINUE *)
IF ( NWORG > IAP ) THEN GOTO 615;
(* TRY THE X POSITION FIRST *)
FOR IBPR := NWORG TO IAP DO
BEGIN
  IF (NOT B[IBPR,1]) THEN GOTO 610
  ELSE
  BEGIN
    IF ((R[IBPR,4]+VEHW) > DECKW) THEN GOTO 610
    ELSE
    BEGIN
      IF ((R[IBPR,3]+VEHL) > DECKL) THEN GOTO 610
      ELSE
      BEGIN
          GOTO 640
      END;
    END;
  END;
610: END;
NWORG := IAP + 1;
615: (* CONTINUE *)
(* SINCE NO MORE X POSITIONS TRY FOR Y *)
IF ( NLORG > IAP ) THEN GOTO 800;
FOR IBPR := NLORG TO IAP DO
BEGIN
  IF (NOT B[IBPR,2]) THEN GOTO 620
  ELSE
  BEGIN
    IF ((R[IBPR,5]+VEHL) > DECKL) THEN GOTO 620
    ELSE
    BEGIN
      IF ((R[IBPR,2]+VEHW) > DECKW) THEN GOTO 620
      ELSE
      BEGIN
          GOTO 645
      END;
    END;
  END;
620: END;
NLORG := IAP + 1;
645: (* CONTINUE *)
NLORG := IBPR + 1;
```

```
ORW := R[IBPR,2];
ORL := R[IBPR,5];
IBPC := 2;
(* FOUND AN ORIGIN NOW GO LOAD VEHICLE *)
GOTO 800;
640: (* CONTINUE *)
NWORG := IBPR + 1;
ORW := R[IBPR,4];
ORL := R[IBPR,3];
IBPC := 1;
(* FOUND AN ORIGIN NOW GO LOAD VEHICLE *)
(* END OF MODULE TO GET ORIGIN *)
800: (* CONTINUE *)
(* SEE IF VEHICLE WILL FIT *)
ORWDW := ORW + VEHW;
ORLDL := ORL + VEHL;
(* IF THIS IS THE FIRST VEHICLE IT MUST FIT, SO LOAD IT NOW *)
IF (IAP <= 0) THEN GOTO 835;
(*IF VEHICLE WILL NOT FIT GO GET ANOTHER ORIGIN, IMPROVE THE
DENSITY IF POSSIBLE.  DO THIS BY FINDING A VEHICLE THAT WILL
INITIALLY FIT AND THEN PROGRESSIVELY MOVE THE VEHICLE TO THE
LEFT,DOWN,  AND TOWARD THE FRONT, IF POSSIBLE. THAT IS, MOVE
THE VEHICLE TOWARD THE FIXED ORIGIN.  REPEAT UNTIL NO FUTHER
IMPROVEMENT IS POSSIBLE IN ANY DIRECTION.*)
SOMCHG := FALSE;
CHECKD := FALSE;
700: (* CONTINUE *)
CHANGE := FALSE;
(*  FIND  MOST RESTRICTING VEHICLE IN LOWER Y  DIRECTION  IN
ORDER TO SEE IF VEHICLE WILL FIT AND TO SEEK AN IMPROVEMENT.
(HOWEVER,  NO  IMPROVEMENT IS POSSIBLE IF ORL IS ALREADY  AT
ITS MIN (0) OR ORIGIN IS AN 'Y' ORIGIN (IBPC = 2). *)
IF (NOT SOMCHG AND (IBPC = 2)) THEN GOTO 711
ELSE
BEGIN
  IF (ORL = 0) THEN GOTO 711
  ELSE
  BEGIN
    SLACK := ORL;
    CHECKD := TRUE;
  END;
END;
FOR I := 1 TO IAP DO
BEGIN
  IF ((R[I,2] >= ORWDW) OR (R[I,4] <= ORW) OR
      (R[I,3] >= ORLDL)) THEN GOTO 710
  ELSE
  BEGIN
    IF (R[I,5] <= ORL) THEN GOTO 705
    ELSE
    BEGIN
```

```
(* GO GET ANOTHER ORIGIN *)
      GOTO 605
    END;
    705: (* CONTINUE *)
    WRITELN('VEHICLE TEST #8');
    XYZ := ORL - R[I,5];
    IF (SLACK > XYZ) THEN SLACK := XYZ
  END;
710: END;
IF (SLACK >= 1) THEN
BEGIN
  CHANGE := TRUE;
  SOMCHG := TRUE;
  ORL := ORL - SLACK;
  ORLDL := ORLDL - SLACK;
END;
711: (* CONTINUE *)
(*  FIND MOST RESTRICTING VEHICLE IN X DIRECTION SIMILAR  TO
ABOVE  SEARCH  MAKE  SURE THE LOCATION IS CHECKED  AT  LEAST
ONCE. *)
IF (NOT CHECKD) THEN GOTO 722
ELSE
BEGIN
  IF (ORW = 0) THEN GOTO 736
  ELSE
  BEGIN
    IF (NOT SOMCHG AND (IBPC = 1)) THEN GOTO 736
  END;
END;
722: (* CONTINUE *)
SLACK := ORW;
CHECKD := TRUE;
FOR I := 1 TO IAP DO
BEGIN
  IF ((R[I,3] >= ORLDL) OR
      (R[I,5] <= ORL) OR (R[I,2] >= ORWDW)) THEN GOTO 730
  ELSE
  BEGIN
    IF (R[I,4] <= ORW) THEN GOTO 725
    ELSE
    BEGIN
      GOTO 605
    END;
    725: (* CONTINUE *)
    XYZ := ORW - R[I,4];
    IF (SLACK > XYZ) THEN SLACK := XYZ
  END;
730: END;
IF (SLACK >= 1) THEN
BEGIN
  CHANGE := TRUE;
```

```
  SOMCHG := TRUE;
  ORW := ORW - SLACK;
  ORWDW := ORWDW - SLACK;
END;
736: (* CONTINUE *)
(* NOW IF ANY CHANGE HAS OCCURED DURING THE ABOVE  SEARCHES,
LOOP  BACK  AND  TRY TO FIND MORE IMPROVEMENT IN  THE  OTHER
DIRECTION *)
IF (CHANGE) THEN GOTO 700;
835: (* CONTINUE *)
(*  THIS  ORIGIN (ORW,ORL) WILL NOT RESTRICT  THE  VEHICLE'S
LOADING. RECORD THE VEHICLE IN ARRAY R *)
 IAP := IAP + 1;
 R[IAP,1] := D[ICP,1];
 R[IAP,2] := ORW;
 R[IAP,3] := ORL;
 R[IAP,4] := ORWDW;
 R[IAP,5] := ORLDL;
 R[IAP,6] := VEHH;
 R[IAP,7] := VSQFT;
 R[IAP,8] := WGT;
 (*  NOW UPDATE LOGICAL ARRAY B (THIS POINT IS  NOW  FILLED)
 AND ESTABLISH THE 2 NEW POSSIBLE ORIGINS *)
 IF (IAP > 1) THEN B[IBPR,IBPC] := FALSE;
 B[IAP,1] := TRUE;
 B[IAP,2] := TRUE;
 (* OVERIDE THESE TRUE SETTINGS OF ORIGINS IF NO VEHICLE CAN
 POSSIBLY FIT *)
 IF ((DECKW - ORWDW) < SMLW) THEN B[IAP,1] := FALSE;
 IF ((DECKL - ORLDL) < SMLL) THEN B[IAP,2] := FALSE;
 (* CUMULATIVE AREA *)
 AREA := AREA + VSQFT;
 GOTO 500;
 999: (* END LOADING PROCESS *)
 ASSIGN(F1,'B:RESULTS.DAT');
 REWRITE(F1);
 FOR I := 1 TO NL DO
 BEGIN
   FOR J := 1 TO MAXCOLR  DO
     F1^[J] := R[I,J];
   SEEKWRITE(F1,I);
 END;
 CLOSE(F1,IOR);
 AFTLOAD(DECKL,DECKW,AREA);
 9999:END; (* LOAD *)

PROCEDURE AFTLOAD(DECKL,DECKW,AREA: INTEGER);
 VAR AA: REAL;

 BEGIN (* AFTLOAD *)
  WRITELN('LOADING IS COMPLETED ... USE THE "DISPLAY"
```

```
                                        OPTION TO GET RESULTS');
  AA := (DECKL /12) * (DECKW/12);
  WRITELN('AREA AVAILBLE WAS ',AA:7:2,' SQUARE FEET');
  WRITELN;
  WRITELN('AREA USED IS ',AREA,' SQUARE FEET');
  SPACEBAR;
 END; (* AFTLOAD *)

MODEND.
```

D. OVERLAY VHELP

```
MODULE OVERLAY23;

EXTERNAL PROCEDURE GOTOXY(X,Y: INTEGER);

EXTERNAL PROCEDURE CLEARSCREEN;

EXTERNAL PROCEDURE SPACEBAR;

(* END OF EXTERNAL DECLARATIONS *)

PROCEDURE HELP;

TYPE STRNG60 = STRING[40];

VAR INFO: ARRAY[1..14] OF STRNG60;
    I: INTEGER;

BEGIN (* HELP *)
  INFO[1] := 'CAVES-COMPUTER AIDED VEHICLE EMBARKATION';
  INFO[2] := 'SYSTEM IS A MENU DRIVEN COMPUTER PROGRAM';
  INFO[3] := 'DESIGNED TO HELP EMBARKATION PERSONNEL  ';
  INFO[4] := 'LOAD VEHICLES ON BOARD A SHIP. TO USE   ';
  INFO[5] := 'CAVES ONE MUST KEY IN ALL PERTINENT DATA';
  INFO[6] := 'ABOUT THE VEHICLES INTO THE VEHICLE FILE';
  INFO[7] := 'THIS WILL INCLUDE THE LENGTH, WIDTH,    ';
  INFO[8] := 'HEIGHT,AREA,WIEGHT AND PRIORITY NUMBER  ';
  INFO[9] := 'OF THE VEHICLE.                         ';
  INFO[10] := 'USE THE "CREATE FILE" OPTION TO CREATE  ';
  INFO[11] := 'THE VEHICLE FILE. ONCE ALL THE VEHICLE  ';
  INFO[12] := 'DATA IS TYPED INTO THE VEHICLE FILE, THE';
  INFO[13] := '"LOAD" OPTION CAN THEN BE UTILIZED TO   ';
  INFO[14] := 'LOAD THE SHIP.                          ';
  CLEARSCREEN;
  GOTOXY(0,0);
  WRITE('DIRECTIONS FOR USE OF CAVES ');
  GOTOXY(0,3);
  FOR I := 1 TO 11 DO
    WRITELN(INFO[I]);
  GOTOXY(0,20);
  SPACEBAR;
END; (* HELP *)

MODEND.
```

LIST OF REFERENCES

1. Bischoff, E. and Dowsland, W. B., "An Application of the Micro to Product Design and Distribution," Journal of Operation Research Society, v. 33, No.3 pp. 271-280, March 1982.

2. Landing Force Training Command, Pacific, Embarkation Reference Phamphlet, Appendix A.

3. Adamowicz, M. and Albano, A., "A Solution of the Rectangular Cutting-Stock Problem," IEEE Transactions on Systems, Man, and Cybernetics, v. SMC-6, No. 2, pp. 302-310, April 1976.

4. Gilmore, P. C. and Gomory, R. E., "The Theory and Computation of Knapsack Functions," Operations Research, v. 14, pp. 1045-1074, November 1966.

5. Ibid..

6. Ingariola, G. and Korsh, J. F., "Reduction Algorithm for 0-1 Single Knapsack Problems," Management Science v. 20, No. 4, pp. 460-463, December 1973.

7. Eilon, S. and Christofides, N., "The Loading Problem," Managament Science, v. 17, No. 5, pp. 259-266, January 1971.

8. New York Scientific Center Report No. 320-2916, On the Optimal Cutting of Defective Glass Sheets, by Susan G. Hahn, October 1967.

9. Christofides, N. and Whitlock, C., "An Algorithm for Two-Dimensional Cutting Problems," Operations Research, v. 25, No. 1, pp. 30-44, January 1977.

10. Steudel, H. J., "Generating Pallet Loading Patterns: A Special Case of the Two-Dimensional Cutting Stock Problem," Management Science v. 25 No. 10 pp. 997-1004, October 1979.

11. Haims, M. J. and Freeman, H., "A Multistage Solution of the Template-Layout Problem," IEEE Transactions on Systems, Science, and Cybernetics, v. SSC-6, No. 2 p. 145, April 1970.

12. Ibid. p. 146.

13. Eilon, S. and Christofides, N. "The Loading Problem," Management Science, v. 17, No. 5, pp. 259-266, January 1971.

14. Hodgson, T. J., "A Combined Approach to the Pallet Loading Problem," IIE Transactions, v. 14, No. 3, pp. 175-182, September 1982.

15. Nelson, B. N., A Container Stuffing Algorithm for Rectangular Solids When Voids May be Required, M.S. Thesis, Naval Postgraduate School, Monterey, California, 1979.

16. Ibid., p. 22.

17. Ibid., p. 24.

18. Ibid., p. 31.

BIBLIOGRAPHY

1. Brown, D. J., Baker, R. S., and Katseff, H. P., "Lower Bounds for On-Line Two-Dimensional Packing Algorithms," Acta Informatica v. 18, 1982.

2. DeSha, E. L., Area-Efficient and Volume-Efficient Algorithms for Loading Cargo, M.S. Thesis, Naval Postgraduate School, Monterey, California, 1970.

3. Eastman, S. E. and Holladay, J. C., Aircraft Loading Considerations: A Sortie Generator for Use in Planning Military Transport Operations, Research Paper P-100, Institute for Defense Analysis, January 1964.

4. Hodgson, T. J., IPLS: Interactive Pallet Loading System, Research Report No. 81-9, Industrial and Systems Engineering Department, University of Florida, Gainesville, Florida, June 1981.

5. Ingargiola, G. and Korsh, J. F., "An Algorithm for the Solution of 0-1 Loading Problems," Operations Research, v. 23, November 1975.

6. Rappe, J. D., Neely, D. P., and Quinn, N. A., The Combat Stores Ship (AFS), M.S. Thesis, Air Force Institute of Technology Air University, 1973.

7. Wang, P. Y., Computational Techniques for Two-Dimensional Rectangular Cutting Stock Problems, Ph.D. Thesis, University of Wisconsin, Milwaukee, 1980.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5100	2
3. Associate Professor Larry Williamson Code 53 Department of Mathematics Naval Postgraduate School Monterey, California 93943-5100	1
4. LtCol Don E. Bonsper Defense Resourses Management Education Center Code 6418 Naval Postgraduate School Monterey, California 93943-5100	1
5. Capt John M. Byzewski P. O. Box 2196 Annapolis, Maryland 21404	3

214097

Thesis
B99
c.1

Byzewski
CAVES - Computer-
aided Vehicle Embark-
ation System.